Leitura e Validação de Dados I

Muitas vezes, um programa interativo requer que sejam lidos e processados vários dados até que um valor que indica o final do programa é lido e, então, o programa é encerrado. Portanto, um programa desta natureza segue o seguinte algoritmo geral:

1. Leia um valor de acordo com as especificações do programa.
2. Enquanto o valor lido é diferente do valor que sinaliza o final do programa faça o seguinte:

   2.1 Processe o valor lido.

   2.2 Leia um valor de acordo com as especificações do programa.
3. Encerre o programa.

A essência daquilo que o programa se propõe a fazer está resumida no passo 2.1 do algoritmo acima, que corresponde ao processamento dos dados recebidos pelo programa. Entretanto, é de suma importância para um bom funcionamento do programa assegurar que os dados introduzidos pelo usuário correspondem às expectativas do programa. Quer dizer, se os dados processados forem inadequados, obviamente, o programa não produzirá resultados significativos.

O tema central deste documento são os passos 1 e 2.2 do algoritmo; ou seja, como ler dados e assegurar que os valores lidos realmente seguem as especificações do programa. Apesar de esta tarefa parecer simples, na maioria das vezes ela não é trivial. A estratégia a ser seguida aqui tem como base apenas o uso da função **scanf()** do módulo de biblioteca stdio. Existem outras abordagens mais sofisticadas (e mais complicadas) para resolver o problema, mas, apesar de apresentar algumas limitações, o método apresentado aqui é satisfatório em muitas situações práticas, além de ser simples e fácil de entender.

Sinalização do Final do Programa

Se o programa encerra quando um determinado valor (por exemplo, 0) é lido, deve-se especificar este valor utilizando uma constante simbólica. Por exemplo:

```
#define VALOR_TERMINAL  0
```

Algumas vezes o programa deve ser encerrado quando um dentre vários possíveis valores é lido. Por exemplo, um programa que encerra quando um valor negativo é lido. Evidentemente, numa tal situação, você não pode especificar todos os valores que causam o encerramento do programa usando constantes simbólicas, mas você não terá dificuldades para adaptar a abordagem descrita aqui para a situação encontrada.

Leitura de Dados com Validação

A função `LeValor()` apresentada a seguir tenta ler valores no meio de entrada padrão usando **scanf()** e retorna um valor apenas quando a entrada é válida de acordo com os critérios do programa. As linhas de interesse da função foram numeradas para facilitar a discussão que segue a apresentação da função.

```
 1. long unsigned LeValor(void)
    {
 2.    long     valor;
 3.    unsigned teste;

 4.    printf("\nIntroduza um numero inteiro positivo: ");
 5.    teste = scanf("%ld", &valor);

 6.    while (!teste || !VerificaValor(valor)) {
 7.       if (teste) {
 8.          printf("\nO valor %ld nao e' valido", valor);
 9.       } else {
10.          printf("\nO valor introduzido nao e' valido");
          } /* if */

11.       printf("\nIntroduza um numero inteiro positivo: ");
12.       LimpaBuffer();
13.       teste = scanf("%ld", &valor);
       } /* while */

14.    LimpaBuffer();

15.    return valor;
    }
```

Comentários sobre a função LeValor()

Nos comentários a seguir, a numeração dos itens a seguir corresponde à numeração de linhas da função `LeValor()`.

1. Aqui, supõe-se que os valores válidos são do tipo **long unsigned**. Substitua o tipo retornado pela função por um tipo de dados adequado ao seu programa.

2. A variável `valor` é utilizada pela função **scanf()** na leitura de valores. Substitua o tipo desta variável por um tipo adequado ao seu programa.

3. A variável `teste` é utilizada para testar o valor retornado pela função **scanf()**. Não é necessário mudar nada aqui.

4. A chamada de **printf()** apresenta uma indicação de como deve ser o valor que o usuário irá introduzir. Modifique-a de acordo com as necessidades de seu programa.

5. Aqui é feita a primeira tentativa de leitura. Substitua o especificador de formato `%ld` por um especificador compatível com o tipo da variável `valor` se o tipo desta variável for modificado em sua definição (linha 2).

6. O corpo desse laço **while** será executado enquanto nenhum valor for lido ou quando o valor lido não satisfizer as especificações do programa. Você precisará modificar a condição do **while** apenas se o teste de validação da entrada for suficientemente simples para substituir a chamada da função `VerificaValor()` (v. adiante).

7. Se a condição do **if** for satisfeita, então um valor foi lido, mas este valor não é válido.

8. Informa ao usuário que a entrada introduzida não válida, apresentando, inclusive, o valor lido. Você deve substituir o especificador `%ld` usado em **printf()** adequadamente.

9. A parte **else** corresponde a um valor introduzido que não é do tipo especificado na chamada de **scanf()** e, portanto, nada foi lido por esta função.

10. Simplesmente, informa ao usuário que a entrada introduzida não é válida. Se preferir, modifique o texto apresentado.

11. A chamada de **printf()** apresenta uma indicação de como deve ser o valor que o usuário irá introduzir. Modifique-o convenientemente para as necessidades de seu programa. Talvez, o programa deva apresentar um prompt mais elaborado desta vez.

12. A função `LimpaBuffer()` esvazia o buffer associado com a entrada padrão (i.e., o teclado). É necessário limpar o buffer de entrada antes de fazer uma nova leitura porque talvez restem caracteres que não foram extraídos pela última chamada da função **scanf()**.

13. Aqui, tenta-se fazer uma nova leitura. Modifique esta instrução convenientemente conforme descrito anteriormente.

14. Novamente, a função `LimpaBuffer()` é chamada para esvaziar o buffer de entrada antes de retornar. Deste modo, garante-se que a próxima leitura (feita pela função `LeValor()` ou qualquer outra função de entrada de dados) irá começar com um buffer limpo.

15. O último valor lido satisfaz os critérios especificados. Logo, este valor é retornado.

Teste dos Valores Lidos

A função `VerificaValor()` apresentada a seguir verifica se o argumento recebido satisfaz os critérios do programa. Se este for o caso, ela retorna 1; caso contrário, ela retorna 0.

```
unsigned VerificaValor(long valor)
{
      return (valor >= 0);
}
```

Comentários sobre a função VerificaValor()

Aqui, supõe-se que o critério que uma entrada válida do programa deve satisfazer é simplesmente que o valor introduzido seja maior do que ou igual a zero. Portanto, a função retorna 1 se o argumento recebido é maior do que ou igual a zero e 0 em caso contrário. Numa situação tão simples quanto esta, você não precisa de uma função de verificação. Isto é, você pode simplesmente fazer o teste de validade do valor lido diretamente na função `LeValor()`. Você pode, por exemplo, substituir, no laço **while** da função `LeValor()`, a condição:

```
(!teste || !VerificaValor(valor))
```

por:

```
(!teste || (valor < 0))
```

Em resumo, a função `VerificaValor()` é recomendada apenas quando a validação dos valores lidos é bem mais complicada do que o que se supõe aqui.

Limpeza do Buffer de Entrada

A função `LimpaBuffer()` lê e descarta todos os caracteres que porventura tenham sido deixados no buffer de entrada em alguma tentativa de leitura de dados. Uma limitação desta função é que, se não houver pelo menos um caractere `'\n'` (i.e., se o usuário não digitou `[ENTER]`), ela irá aguardar a introdução deste caractere antes de liberar a entrada. Portanto, antes de chamar esta função certifique-se que há realmente algum caractere no buffer de entrada.

```
void LimpaBuffer(void)
{
    (void) scanf("%*[^\n]");
    (void) getchar();
}
```

A primeira instrução da função `LimpaBuffer()` lê e descarta (*descartar* corresponde a `*` no especificador de formato) todos os caracteres encontrados no buffer de entrada até que o caractere `'\n'` seja encontrado (`[^\n]` no especificador de formato corresponde a ler todos os caracteres até que `'\n'` seja encontrado). O próprio caractere `'\n'`, entretanto, não é lido. Portanto, a chamada de **getchar()** é necessária para completar a limpeza (i.e., ler o caractere `'\n'`).

Os operadores de *casting* **(void)** foram utilizados nas duas instruções da função `LimpaBuffer()` apenas por uma questão de legibilidade. Isto é, para indicar que as funções **scanf()** e **getchar()** retornam valores que se preferiu não usar (obviamente, por não serem necessários neste caso).

A função `LimpaBuffer()` apresentada aqui realiza exatamente a mesma tarefa que aquela apresenta na **Seção 2.6**. A diferença entre estas funções está na forma de implementação.

Chamada da Função de Leitura

A função **main()** a seguir lê valores apropriados para o programa e processa-os repetidamente até que um valor que sinaliza o final do programa é lido. As linhas de interesse da função foram numeradas para facilitar a discussão que segue a apresentação da função.

```
    int main(void)
    {
1.     long unsigned valorLido;

2.     valorLido = LeValor();

3.     while(valorLido != VALOR_TERMINAL) {
4.        printf("\nO valor lido foi: %ld\n", valorLido);
5.             /*** Processe aqui o valor lido ***/
6.        valorLido = LeValor();
       }

       return 0;
    }
```

Comentários sobre a função main()

1. A variável `valorLido` armazena o mais recente valor válido lido pelo programa. Aqui, supõe-se que os valores válidos são do tipo **long unsigned**. Substitua este tipo pelo tipo adequado ao seu programa.

2. Aqui, é feita uma leitura e validação de entrada introduzida pelo usuário por meio da função `LeValor()` descrita anteriormente.

3. O corpo deste laço **while** será repetido enquanto não se encontra um valor, especificado pela constante `VALOR_TERMINAL`, que indica o término do programa. Se houver vários valores que sinalizem o término do programa, modifique esta condição adequadamente. Por exemplo, se o programa deve terminar quando for lido um valor maior do que 20, modifique a condição do **while** para:

```
while(valorLido <= 20)
```

   Neste último exemplo, melhor ainda seria se a condição fosse escrita como: `valorLido <= VALOR_LIMITE`, onde `VALOR_LIMITE` é uma constante definida usando **#define**.

4. Esta instrução informa ao usuário qual foi o valor mais recentemente lido. Embora esta informação pareça óbvia e redundante para ser apresentada ao usuário, devido às limitações da abordagem descrita aqui, é necessário que o usuário seja notificado sobre que valor está sendo processado. Por exemplo, se o usuário digitar (acidentalmente) como entrada `2O` (leia-se dois seguido de ó e não dois seguido de zero), a função `LeValor()` irá retornar 2 como sendo o valor válido lido (talvez, o usuário esperasse ter introduzido vinte).

5. Este é o espaço reservado para processamento do valor lido. Introduza aqui as instruções necessárias para este processamento, se o mesmo for simples ao ponto de não prejudicar a legibilidade do programa, ou uma chamada de função que realiza o processamento desejado se este processamento tiver um certo grau de complexidade. A apresentação de resultados do processamento pode ser feita neste espaço, logo após o processamento propriamente dito, quando o processamento de cada valor produz individualmente um resultado (e.g., o cálculo da raiz quadrada de cada valor), ou então logo antes da instrução **return** que encerra o programa, quando o conjunto de valores como um todo produz o resultado desejado (e.g., o cálculo da média aritmética de todos os valores introduzidos).

6. Esta instrução lê o próximo valor conforme descrito no comentário da linha 2. O laço **while** é, então, recomeçado.

Limitações do Método Utilizado

O método de leitura e validação de dados apresentado acima contém outras limitações, além daquelas já indicadas aqui. Estas limitações serão comentadas e superadas num método similar, mas mais poderoso, apresentado na **Seção 10.9**.